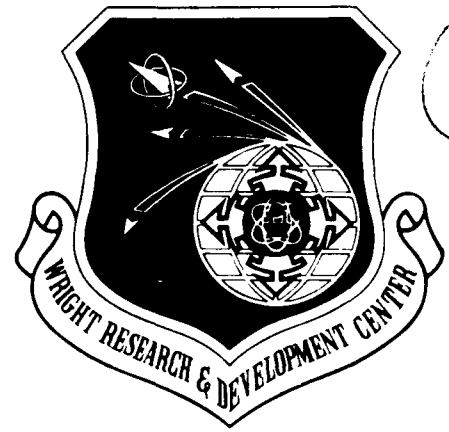


WRDC-TR-90-8007
Volume V
Part 48

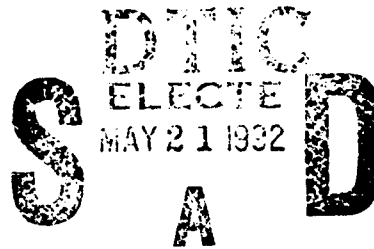
AD-A250 479



INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume V - Common Data Model Subsystem
Part 48 - Embedded SQL Reference Manual

K. Stephey, J. Slaton

Control Data Corporation
Integration Technology Services
2970 Presidential Drive
Fairborn, OH 45324-6209



September 1990

Final Report for Period 1 April 1987 - 31 December 1990

Approved for Public Release; Distribution is Unlimited

92-13523



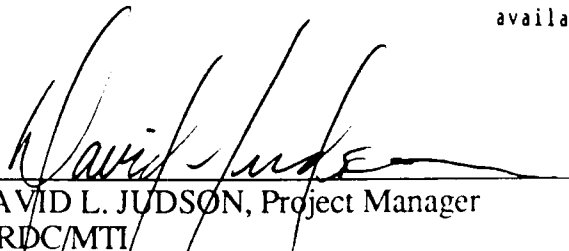
MANUFACTURING TECHNOLOGY DIRECTORATE
WRIGHT RESEARCH AND DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6533

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, regardless whether or not the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data. It should not, therefore, be construed or implied by any person, persons, or organization that the Government is licensing or conveying any rights or permission to manufacture, use, or market any patented invention that may in any way be related thereto.

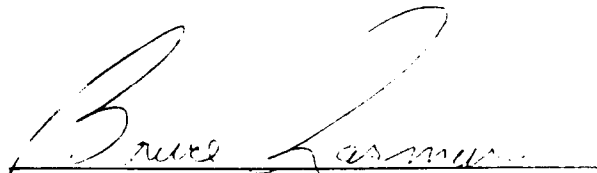
This technical report has been reviewed and is approved for publication.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations


DAVID L. JUDSON, Project Manager
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

FOR THE COMMANDER:


BRUCE A. RASMUSSEN, Chief
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WRDC/MTI, Wright Patterson Air Force Base, OH 45433-6533 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution is Unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) PRM620341440		5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR-90-8007 Vol. V, Part 48	
6a. NAME OF PERFORMING ORGANIZATION Control Data Corporation; Integration Technology Services	6b. OFFICE SYMBOL (if applicable) WRDC/MTI	7a. NAME OF MONITORING ORGANIZATION WRDC/MTI	
6c. ADDRESS (City, State, and ZIP Code) 2970 Presidential Drive Fairborn, OH 45324-6209		7b. ADDRESS (City, State, and ZIP Code) WPAFB, OH 45433-6533	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Wright Research and Development Center, Air Force Systems Command, USAF	8b. OFFICE SYMBOL (if applicable) WRDC/MTI	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUM. F33600-87-C-0464	
8c. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB, Ohio 45433-6533		10. SOURCE OF FUNDING NOS.	
11. TITLE (In See block 19		PROGRAM ELEMENT NO. 78011F	PROJECT NO. 595600
		TASK NO. F95600	WORK UNIT NO. 20950607
12. PERSONAL AUTHOR(S) Control Data Corporation: Stephey, K., Slaton, J.			
13a. TYPE OF REPORT Final Report	13b. TIME COVERED 4 / 1 / 87 - 12 / 31 / 90	14. DATE OF REPORT (Yr., Mo., Day) 1990 September 30	15. PAGE COUNT 42
16. SUPPLEMENTARY NOTATION WRDC/MTI Project Priorit 6203			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify block no.)	
FIELD	GROUP	SUB GR.	
130E	0905		
19. ABSTRACT (Continue on reverse if necessary and identify block number) This manual lists and describes SQL commands that are embedded within application programs to retrieve information described to the Common Data Model. BLOCK 11: INTEGRATED INFORMATION SUPPORT SYSTEM Vol V - Common Data Model Subsystem Part 48 - Embedded SQL Reference Manual			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED x SAME AS RPT. DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL David L. Judson	22b. TELEPHONE NO. (Include Area Code) (513) 255-7371	22c. OFFICE SYMBOL WRDC/MTI	

FOREWORD

This technical report covers work performed under Air Force Contract F33600-87-C-0464, DAPro Project. This contract is sponsored by the Manufacturing Technology Directorate, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Bruce A. Rasmussen, Branch Chief, Integration Technology Division, Manufacturing Technology Directorate, through Mr. David L. Judson, Project Manager. The Prime Contractor was Integration Technology Services, Software Programs Division, of the Control Data Corporation, Dayton, Ohio, under the direction of Mr. W. A. Osborne. The DAPro Project Manager for Control Data Corporation was Mr. Jimmy P. Maxwell.

The DAPro project was created to continue the development, test, and demonstration of the Integrated Information Support System (IISS). The IISS technology work comprises enhancements to IISS software and the establishment and operation of IISS test bed hardware and communications for developers and users.

The following list names the Control Data Corporation subcontractors and their contributing activities:

<u>SUBCONTRACTOR</u>	<u>ROLE</u>
Control Data Corporation	Responsible for the overall Common Data Model design development and implementation, IISS integration and test, and technology transfer of IISS.
D. Appleton Company	Responsible for providing software information services for the Common Data Model and IDEF1X integration methodology.
ONTEK	Responsible for defining and testing a representative integrated system base in Artificial Intelligence techniques to establish fitness for use.
Simpact Corporation	Responsible for Communication development.
Structural Dynamics Research Corporation	Responsible for User Interfaces, Virtual Terminal Interface, and Network Transaction Manager design, development, implementation, and support.
Arizona State University	Responsible for test bed operations and support.

TABLE OF CONTENTS

			<u>Page</u>
SECTION	1.	INTRODUCTION	1-1
SECTION	2.	SYSTEM OVERVIEW	2-1
SECTION	3.	SQL COMMANDS	3-1
	3.1	Data Retrieval Commands.....	3-2
	3.1.1	Queries Which Return A Single Row.....	3-2
	3.1.2	Queries Which Return Multiple Rows	3-3
	3.1.3	Syntax Explanation.....	3-3
	3.1.4	Comments.....	3-4
	3.2	DELETE Command.....	3-16
	3.3	INSERT Command	3-18
	3.4	UPDATE Command	3-20
	3.5	Transaction Commands.....	3-22
	3.5.1	ROLLBACK Command	3-22
	3.5.2	COMMIT Command	3-22
	3.6	Cursor Commands.....	3-22
	3.6.1	Declare Cursor command	3-23
	3.6.2	OPEN CURSOR command	3-23
	3.6.3	FETCH command	3-23
	3.6.4	CLOSE CURSOR command.....	3-24
	3.7	Distributed Update Restrictions.....	3-25
	3.8	Error Codes	3-26
SECTION	4.	EMBEDDING SQL IN A PROGRAM.....	4-1
	4.1	COBOL Syntax.....	4-1
	4.2	FORTRAN Syntax.....	4-1
	4.3	C Syntax.....	4-1
APPENDIX	A	BACKUS-NORMAL FORM OF SQL	A-1
APPENDIX	B	EXTERNAL SCHEMA DATA TYPES	B-1

SECTION 1

INTRODUCTION

The Embedded SQL used by the Common Data Model (CDM) is adopted from the American National Standard Institute (ANSI) standard, numbered X3H2-89-27 dated November 16, 1988. The Data Manipulation Language specification portion of the ANSI SQL standard is used by the CDM as the data manipulation language for the access to databases in a heterogeneous, distributed database management system (DBMS) environment. The Embedded SQL used by the CDM is a subset of the full SQL Data Manipulation Language specification.

The CDM Embedded SQL is used as embedded statements in the host languages of COBOL, FORTRAN, and C.

The CDM precompiler is used to process the application program containing embedded SQL statements before the host language compiler is used.

The important property of the Embedded SQL to keep in mind when using this manual is that you perceive all data to be in the form of tables. Data within the database can be considered to be stored as tables even if containing only one row of values. Similarly, only tables can be retrieved from the database, even if the table consists of a single "row" with a single "column" (i.e., only a single value). This important property of relational databases allows the output of one retrieval command to be utilized as the input to another operation without worrying about the structure of the data. Furthermore, data can be retrieved and used without having to specify the structure of the data for each application and the size of the data.

Tables are usually called "relations" and the terms table and relation will be used synonymously here. Similarly, rows of the table may be called "records" or "tuples" and columns may be called "data fields," "data items" or "attributes." An individual number or character string entry in the table will be called a "value".

Each of the following sections on specific commands begins with the syntax of the command. The syntax is presented using a method that is described at the beginning of Section 3; it is similar to the method used in the Neutral Data Definition Language (NDDL) Reference Manual. The rigorous BNF description of the language is presented in Appendix A. Following the syntax of the command, semantic notes point out commands and restrictions. This document should provide sufficient information for an application programmer to begin work. Moreover, you will find it an appropriate reference in the future when you have become familiar with Embedded SQL.

If you are unfamiliar with SQL, you should consult tutorials and references on that language before using this guide.

Accession For	
NTIS	CRA&I
DTIC	TAB
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution	
Availability Codes	
Dist	Availability or Special
A-1	

References include:

Chamberlin, D.D., et al., "Sequel 2: A Unified Approach to Data Definition, Manipulation, and Control," IBM Journal of Research and Development. Vol 20, No. 6, Nov. 1976, pp. 560-575.

Date, C.J., A Guide to DB2. Addison-Wesley Publ. Co., 1984.

In addition, many commercial relational database systems offer interface languages similar to SQL. The manuals for these languages are useful for becoming acquainted with the general syntax of SQL.

SECTION 2

SYSTEM OVERVIEW

The processing system is known as the Common Data Model precompiler (CDMP). The precompiler provides the application programmer with important capabilities to:

- Request database accesses in a non-procedural data manipulation language (SQL) that is independent of the data manipulation language (DML) of any particular data base management system.
- Request database access using SQL that specifies accesses to a set of related records, rather than to individual records (i.e., using a relational DML).
- Request access to data that are distributed across multiple databases with a single SQL command, with minimum knowledge of data locations or distribution details.

Information about external schemas, the conceptual schema and internal schemas (including data locations) is provided by CDMP access to the Common Data Model (CDM) database. The CDM is a relational database of metadata pertaining to CDM. It is described by the CDM1 information model using IDEF1X. The precompiler parses the application program source code, identifying SQL commands. It applies external-schema to conceptual-schema and conceptual-schema to internal-schema transforms on the SQL command, thereby decomposing the SQL command into internal-schema, single database requests. These single database requests are each transformed into generic data manipulation language (DML) commands. Programs are generated from the generic DML commands which can access the specific databases to accomplish the request. These programs, referred to as Request Processors (RP), are stored at the appropriate host machines. The SQL commands in the application source program are replaced by host-language code which, when executed, activates the run-time request evaluation processes associated with the particular SQL command.

The precompiler also generates a Conceptual Schema/External Schema (CS/ES) Transformer program which will take the final results of the request, stored in a file as a table with external-schema structure, and convert the data values into the correct form for presentation. The CS/ES Transformer also performs SQL function operations on the data.

Finally, the precompiler generates a Join Query Graph and Result Field Table which are used by the Distributed Request Supervisor (DRS) during the run-time evaluation of the SQL request.

The DRS is responsible for coordination of the run-time activity associated with the evaluation of an SQL command. It is activated by the application program, which sends it the names and locations of the request processors to activate along with run-time parameters which are to be sent to them. The results generated by the request processors are stored as files in the form of conceptual-schema relations on the host which executed the request process. Using the Join Query Graph, transmission cost information and data about intermediate results, the DRS determines the optimal strategy for combining the intermediate results of the SQL command. It issues the appropriate file transfer request, activates aggregators to perform unions, joins, and NOT IN SET operations, and activates the appropriate CS/ES Transformer program to transform

the final results. Finally, the DRS notifies the application program that the request is completed, and sends it the name of the file which contains the results of the request.

SECTION 3

SQL COMMANDS

The following conventions are used in the description of the SQL commands at the beginning of the following subsections:

3.1 Conventions

3.1.1 Notation

UPPER CASE WORDS denote keywords in the command

LOWER CASE WORDS denote user-defined words

{ } denotes that exactly one of the options within the braces must be selected by the user

• • • denotes repetition of the last element

[] denotes that the entry within the brackets is optional

| denotes an "or" relationship among the entries

_ denotes default option

3.1.2 Punctuation

1. A "." is used to separate the table-label (i.e., table alias) from the column-name. The table-label is used to match a column to a specific table in the list of tables referenced in the FROM clause.
2. A ":" is placed before the name of a host-language program variable that will receive returned values.
3. A "," is inserted between entries in lists of clauses.
4. Parentheses are used to enclose the column-list in an INSERT statement.
5. Parentheses are used to enclose the object column of a function.
6. Parentheses are used to enclose the values to be inserted in an INSERT statement.
7. A set of parentheses of group logical conditions in the WHERE clause may be nested.
8. A set of parentheses may be used to group combinations of SELECT statements. They may be nested.

3.1.3 Character Case

Keywords are not case sensitive. User-defined words are case sensitive.

3.1.4 Word Length

Table labels are limited to 2 characters.

Table and column names are defined by the relational view in use.

3.1.5 SQLCODE

SQLCODE is a generic term used in this document to indicate SQL error status.

For language specific implementation, refer to Section 3.8.

3.2 Data Retrieval Commands

3.2.1 Queries Which Return a Single Row

For queries which will return a single row, the syntax is as follows:

NOTE: a cursor is not used

```
ORDER BY cannot be used
SELECT  [DISTINCT|ALL] {expr-spec,...}
        { *          }

      INTO    [:]variable-name [[INDICATOR:] ind-var-name],...

      FROM    table-name [table-label], ...
      [WHERE  predicate-spec
      [AND    predicate-spec ...]]
```

3.2.2 Queries Which Return Multiple Rows

For queries which may return more than one row of data, you must use the DECLARE CURSOR, SELECT, FETCH, OPEN CURSOR and CLOSE CURSOR statements.

The syntax is as follows:

```
DECLARE      cursor-name  CURSOR FOR

      {SELECT  [DISTINCT|ALL]      {expr-spec,...}
                                     { *      }

      FROM      table-name [table-label], ...
      [WHERE    predicate-spec
      [AND      predicate-spec ...]]

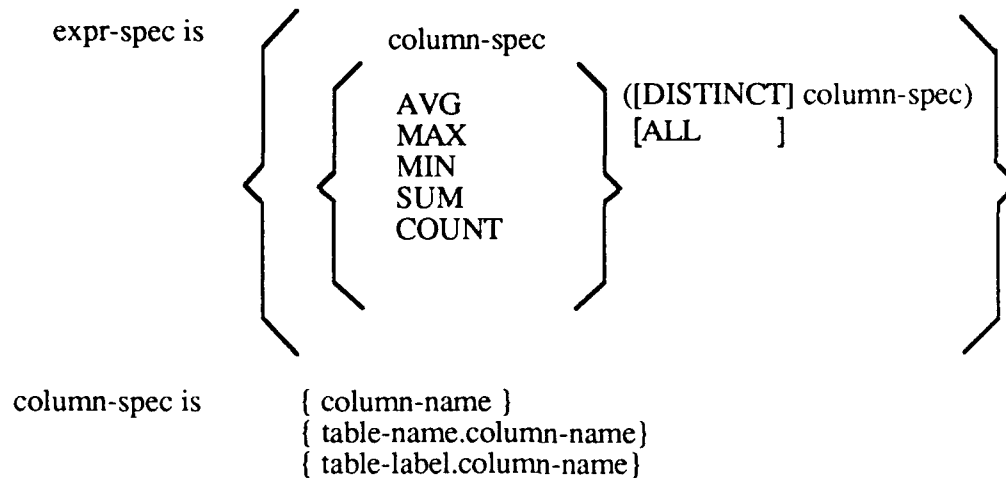
      [( {UNION}  [ALL]
          {MINUS}
          {INTERSECT}

      SELECT [DISTINCT|ALL] {expr-spec,...}
                                     { *      }
      FROM      table-name [table-label],...
      [WHERE    predicate-spec
      [AND      predicate-spec ...] ] }...]

      [ORDER BY {unsigned integer} [ASC|DESC], ...]
                  {column-spec      }
```

3.2.3 Syntax Explanations

- variable-name and ind-var-name are defined in the host program, preceding colons must be used on both the variable-name and ind-var-name or not at all.
- table-label is a one- or two-character name
- table-name and column-spec are defined for the relational view
- [*] designates all columns for the table in the order defined in the relational view
- predicate-spec is either a column-join-, like-, between-, or null-predicate (see Appendix A)
- cursor-name is a name unique among all other cursors declared within the host program
- unsigned-integer is the ordinal position of the column-spec
- If more than one SELECT statement is specified in a DECLARE CURSOR statement and the ORDER-BY clause is used, the unsigned integer option of the ORDER-BY clause must be used.



3.2.4 Comments

(a) SELECT Keyword

The SELECT command is the only command used in SQL to retrieve data from the distributed database.

(b) INTO Variable Assignments

The data retrieved by a SELECT command is assigned to program variables using a variable-assignment construct.

If an indicator variable was specified for a column, a value will be placed in the indicator variable on retrieval indicating whether or not the column was null. A value of 1 in the indicator variable indicates a null column, and 0 means not null. The following indicator variable definitions should be used for each language:

COBOL	PIC	9
FORTRAN	CHARACTER	*1
C	char	

A variable name should not be tested for null, only test indicator variables. The results in a variable name for a null column are dependent on the particular DBMS the results are being retrieved from.

The null field itself will contain the null value specified in the NDDL DEFINE DATABASE command.

If not selecting statistical functions, refer to Appendix B for proper result variable definitions. If selecting statistical functions, refer to Section 3.1.4 (e) for result variable definitions.

The following are examples of valid SELECT statements.

- ex. A. SELECT D.DNO, D.DNAME, D.DLOC, D.DSIZE
 INTO :DEPTNO, :DEPTNAME, :DEPTLOC, :DEPTSIZE
 INDICATOR: IND1
 FROM DEPT D
- ex. B. DECLARE CUR1 CURSOR FOR
 SELECT D.DNO D.DNAME D.DLOC D.DSIZE
 FROM DEPT D
- ex. C. SELECT DEPT.DNO, D.DNAME, D.DLOC, D.DSIZE
 INTO :DEPTNO, :DEPTNAME, DEPTLOC, DEPTSIZE
 FROM DEPT D
 WHERE D.DLOC != 'LAX'
- ex. D. DECLARE CUR1 CURSOR FOR
 SELECT DEPT.DEPT_NO
 FROM DEPT
 WHERE DEPT_NAME = 'ACCOUNTING'
 UNION
 (SELECT E.DNO
 FROM EMP E
 WHERE E.EMPNO > 100
 INTERSECT
 SELECT M.DEPT-NO
 FROM MGR M)
 ORDER BY 1 ASC

(c) SELECT DISTINCT Phrase

The DISTINCT clause on a SELECT statement is used to specify that duplicate rows are to be removed prior to presentation of the results. Omitting the DISTINCT clause implies that duplicate rows are not removed unless you specified this clause at NDDL Create View time. If the DISTINCT clause is specified in both the SQL select and the NDDL CREATE VIEW, the result is the same as if it had been specified on only one statement.

The DISTINCT phrase refers to the entire set of selected columns following it. For example, SELECT DISTINCT * FROM T1 removes only those rows from T1 for which all column values are identical to those of another row in T1. The DISTINCT processing is applied to rows in their external-schema formats.

```
SELECT DISTINCT * INTO VAR1, VAR2, VAR3
FROM DEPT D
WHERE D.LOC = 'LAX'
```

```
DECLARE DEPT-CUR CURSOR FOR
SELECT DISTINCT
D.DNO D.DNAME D.LOC
FROM DEPT D
WHERE D.SIZE = 'LARGE'
```

(d) Statistics Functions

Function expressions can be presented as the result of a SELECT statement only; they cannot be used in a WHERE or ORDER BY clause. These functions are used to specify that column statistics of AVG value, MAX value, MIN value, SUM value, or COUNT of rows are to be produced. The ORDER BY clause should not be used when functions are specified because unnecessary processing will be performed. A SELECT DISTINCT specification should not be used with functions because it causes unnecessary processing.

The results of AVG (column) are the same as the results of SUM(column)/COUNT(column). All values are considered unless the optional DISTINCT phrase within the function clause is included; in which case, duplicate values are removed prior to the function application. All statistical functions ignore nulls in the data.

SELECT cannot return both a table and the result of functions in a single statement. Thus, if one function is specified in an expr-spec, then all values to be retrieved must be the one result of functions. It is permissible to retrieve several functions, but the user should be aware that the values in the single row returned will not necessarily have any logical relationship.

MIN, MAX and COUNT can be applied to both numeric and string columns. Specification of function DISTINCT before MIN or MAX is ignored. AVG and SUM can be applied only to numeric columns. Functions are applied to columns in their external-schema formats. For the empty set, COUNT returns zero and other functions return an undefined result; the existence of the empty set for non-COUNT functions results in a condition code set in SQLCODE, as discussed below.

When declaring result variables for SUM, AVG, and COUNT, the following formats should be used:

	<u>COBOL</u>	<u>FORTRAN</u>	<u>C</u>
SUM	S9(9)V9(9)	DOUBLE PRECISION	double
AVG	S9(9)V9(9)	DOUBLE PRECISION	double
COUNT	9(9)	INTEGER	long

When declaring result variables for MIN and MAX, the variables should be defined according to the datatype of the column being selected. Refer to Appendix B for the proper definitions.

If a function operates on an empty column, a result may be returned that is not valid (for example, SUM will return 0). The SQLCODE flag (and associated null indicator if used) should be checked by the application program before using the result returned by a function.

```
SELECT
    AVG(P.LEAD-TIME), MIN(P.LEAD-TIME), MAX(P.LEAD-TIME),
    INTO :LEAD-AVG INDICATOR :IND1,
        :LEAD-MIN INDICATOR :IND2,
        :LEAD-MAX INDICATOR IND3
FROM PART P WHERE P.SIZE > 100
```

```
DECLARE COUNTCUR CURSOR FOR
SELECT COUNT(D.DNAME)
FROM DEPT D, EMP E
WHERE E.DNO = D.DNO
```

```
DECLARE CUR1 CURSOR FOR
SELECT
    MIN(SE.SALARY) MIN(HE.RATE)
FROM SALARIED-EMP SE, HOURLY-EMP HE
WHERE SE.EMP-NO = HE.EMP-NO
```

```
SELECT COUNT(DISTINCT E.JOB)
    INTO COUNT-JOB INDICATOR COUNT-IND
FROM EMP E
WHERE E.DNO = 10
```

```
SELECT COUNT(DISTINCT D.LOC)
    INTO :VAR1 :IND1
FROM DEPT D, NEWDEPT N
WHERE D.NAME = N.DNAME
```

Note: User-defined functions and explicit arithmetic functions (e.g., WEIGHT * 2.2) are *not* supported.

(e) FROM Clause

Table labels are not required. If two or more tables are specified in the table-list, it is a good idea to be concise and use table labels or table names to designate columns.

(f) WHERE Clause

The WHERE clause is used to limit the information returned from one or more tables. If the WHERE clause is not specified and one table is specified in the FROM clause, all rows from the table are returned. If the WHERE clause is not specified and more than one table is specified, unpredictable results may occur, i.e., cross product.

Only column-predicate or join-predicate comparisons are allowed in WHERE clauses. The column-predicate compares the value of a column with a single specific value indicated by the contents of a scalar program variable, a literal string in quotes, or a number. Either the column name or value can be the first object of the comparison (only the case in which the value is second is shown in the syntax above). The join-predicate is an entity to entity join (equi-join, outer-join,

and no-in-set) using one column from each entity. AND clauses can be used to specify multiple qualifications on the table selected. The comparison operator (bool-op) includes most common operations (including parenthesis, NOT, AND, OR, XOR, Boolean operators, Between and Null operators), but does not include an "IN," "EXISTS," "ALL," or "ANY" comparison that would allow a column to be compared with many values. If there are any join-predicate comparisons in the WHERE clause, they must all be listed first or all listed last. They cannot be interspersed with the column-predicate comparisons. They also must only be "ANDed" together, not "ORed."

The qualifications specified in the WHERE clause of an SQL statement will be "ANDed" with those specified in the WHERE clause of the CREATE VIEW. These qualifications include the join-predicates, column-predicates, and bool-ops. If a SELECT has no WHERE clause, but there is a WHERE clause on the view, the view WHERE clause will still be enforced. A description of the CREATE VIEW command may be found in the NDDL Reference Manual.

The join-predicate comparison allows only the equi-join, outer-join, and not-in-set operations; the operators <, <=, > and >= are not implemented. The join fields compared in a join-predicate need not have identical data types in the user's (external) view of the table, except that numeric data must be compared with numeric data and character strings with character strings.

The equi-join is specified by placing an "=" between two columns. The equi-join connects a row from each of two tables to form one row in the result table if the values in the specified columns in the tables are identical. Duplicate rows will be returned if duplicate rows exist in either table. Rows for which a match is not found are not included in the result table.

The not-in-set is specified by placing an "<>" or an "!=" between two columns. The not-in-set connects a row from each table to form one row in the result table if the values in the specified columns in the tables are not identical. Duplicate rows will be returned if duplicate rows exist in either table. Rows for which a match is found are not included in the result table.

The outer-join is specified by placing an "=" between two columns and an "(+)" after the second column. The outer-join operation is a selection procedure that is similar to join, but when rows from the table specified to the left of the = operator do not match entries in the table to the right, a "partial" results row will be considered for retrieval. For a more detailed explanation, refer to C.J. Date, An Introduction to Database Systems, 3rd Edition. Because columns from the table on the right may have been selected, null values may be introduced. The query may specifically exclude or include those "partial" rows by use of the IS[NOT] NULL predicate applied to one of the columns from the right table. For example, with the following request:

```
SELECT D.DNO, D.DNAME, E.NAME
      INTO :DEPTNO, :DEPTNAME, :EMPNAME
      FROM DEPT D, EMP E
      WHERE D.DNO = E.DNO (+)
```

if the following data is found,

<u>D.DNO</u>	<u>E.DNO</u>
1	2
2	3
4	5
5	6
7	8
8	9

the result will have "full" rows (without a null E.ENAME) for

<u>D.DNO</u>
2
5
8

and "partial" rows (null E.NAME) for

<u>E.DNO</u>
1
4
7

since these departments did not have employees.

In the example below, the outer-join is applied to the employee table. This is useful often in validating the subset constraints (entity dependence) defined in the conceptual schema.

```
SELECT E.DNO, E.ENAME,  
       D.DNO, D.DNAME  
       INTO :EDEPTNO, :EMPNAME, :DEPTNO, :DEPTNAME  
FROM DEPT D , EMP E  
   WHERE E.DNO = D.DNO (+)
```

The result will have "full" rows for

<u>E.DNO</u>
2
5
8

and "partial rows" row for

<u>E.DNO</u>
3
6
9

Some columns cannot be specified in a WHERE clause because the column in the conceptual schema maps to non-normalized database structures in the internal-schema databases. In particular, a conceptual-schema column that maps to a data field in a repeating group in the internal database will not have a unique value for each row. The precompiler will recognize this problem and reject the SQL request. The CDM Administrator (CDMA) should inform the user of these restrictions before precompilation. The CDMA can determine those by examining conceptual-internal schema mapping relationships.

(g) Logic Rules for WHERE clause

NOT

The NOT operator will be translated according to DeMorgan's Law: Operators are reversed, AND becomes OR and OR becomes AND.

BETWEEN

A column can be compared to other columns of the same table, literal values, numeric constants, or program variables. The statement translates as follows:

WHERE A.X BETWEEN 7 AND :VAR-X

will be translated to

WHERE (A.X >= 7 AND A.X <= :VAR-X)

and

A.X NOT BETWEEN 'AAA' AND 'KKK'

will be translated to

(A.X < 'AAA' OR A.X > 'KKK').

BETWEEN is understood to be inclusive of the end points.

NOT EQUAL

Note that both <> and != are allowed for inequality tests.

LIKE

The following rules have been derived or extracted from the ANSI SQL standard X3H2-89-27 of November 16, 1988.

(g1) Let x denote the value referenced by the <column specification> and let y denote the result of the <value specification> of the <pattern>.

(g2) Case:

* If an <escape character> is specified, then:

- Let z denote the result of the <value specification> of the <escape character>.

- There shall be a partitioning of the string y into substrings such that each substring is of length 1 or 2, no substring of length 1 is the escape character z, and each substring of length 2 is the escape character z followed by either the escape character z, an underscore character, or the percent sign character. In that partitioning of y, each substring of length 2 represents a single occurrence of the second character of that substring. Each substring of length 1 that is the underscore character represents an arbitrary character specifier. Each substring of length 1 that is the percent sign character represents an arbitrary string specifier. Each substring of length 1 that is neither the underscore character nor the percent sign character represents the character that it contains.

* If an <escape character> is not specified, then each underscore character in y represents an arbitrary character specifier, each percent sign character in y represents an arbitrary string specifier, and each character in y that is neither the underscore character nor the percent sign character represents itself.

(g3) The string y is a sequence of the minimum number of substring specifiers such that each <character> of y is part of exactly one substring specifier. A substring specifier is an arbitrary character specifier, an arbitrary string specifier, or any sequence of <character>s other than an arbitrary character specifier or an arbitrary string specifier.

(g4) "x LIKE y" is unknown if x or y is the null value. If x and y are nonnull values, then "x LIKE y" is either true or false.

(g5) "x LIKE y" is true if there exists a partitioning of x into substrings such that:

* A substring of x is a sequence of zero or more contiguous <character>s of x and each <character> of x is part of exactly one substring.

* If the i-th substring specifier of y is an arbitrary character specifier, the i-th substring of x is any single <character>.

* If the i-th substring specifier of y is an arbitrary string specifier, the i-th substring of x is any sequence of zero or more <character>s.

* If the i-th substring specifier of y is neither an arbitrary character specifier nor an arbitrary string specifier, the i-th substring of x is equal to that substring specifier and has the same length as that substring specifier.

* The number of substrings of x is equal to the number of substring specifiers of y.

(g6) "x NOT LIKE y" has the same result as "NOT (x LIKE y)".

(h) ORDER BY Clause

The ORDER by clause is used to specify the sequencing rules for presentation of the results of a SELECT operation. Omitting the ORDER by clause on a SELECT statement implies that the rows of the result table are presented in a system-determined order.

The columns in the order-spec-list control the sorting of result rows in major-to-minor order. If the direction phrase is omitted for a column, then ASC (ascending) is assumed. The columns of an order-spec-list need not all have the same accompanying direction.

The items to be ordered by can be specified either by column name or by an unsigned integer. If more than one SELECT appears in the DECLARE CURSOR statement, then unsigned integers must be used. The unsigned integer refers to the ordinal position of the column in the result table. If column names are used, the columns specified do not have to be ones appearing in the column-list of the SELECT phrase.

Sorting is done on the columns in their external-schema formats and will be done on the machine running the application program. The order of the sorted result will depend on the storage code used by the computer running the applications program. ASCII is to be used whenever possible. Thus, the result of the same program can differ if it is precompiled and run on different

machines. Null values are treated as the largest representation and will appear last when the ASCENDING option is chosen and first when the DESCENDING option is chosen.

```
DECLARE CUR1 CURSOR FOR
SELECT E.NAME, E.DEPT, E.PHONE
FROM EMP E
WHERE E.JOB CODE > 50
ORDER BY E.NAME
```

```
DECLARE CUR2 CURSOR FOR
SELECT PART#, SIZE
FROM PART
ORDER BY 2 DESCENDING
```

```
DECLARE CUR3 CURSOR FOR
SELECT D.DEPT#, D.LOC, D.CITY
FROM DEPT D
ORDER BY D.CITY ASC, D.LOC DESC, D.SIZE ASC
```

(i) Nulls

Nulls are intentionally introduced into a query result by use of the outer join along with selection of columns from the second table when the data does not match. These can be tested for with the IS [NOT] NULL predicate. A value to be recognized as NULL for the internal schema databases will be stored in the CDM. These shall be used when qualifying (use of IS [NOT] NULL) on an external schema data item that maps to this internal schema value, outside of the outer join application.

To allow testing for nulls of retrieved data values, use the optional INDICATOR clause. The ind-var-name is set to 1 if a null value is encountered. If a null value is found, the value of the variable-name is unreliable because it is dependent on the particular DBMS that value was retrieved from.

For example,

```
SELECT A.B, A.C, A.F
INTO :BVAR INDICATOR: BVAR-FLAG,
      CVAR CVAR-FLAG,
      FVAR INDICATOR FVAR-FLAG
FROM TABLE1 A
```

(j) Grouping Clauses

This release does not support GROUP BY and HAVING clauses to determine aggregate properties of multiple rows of a table. These operations must be performed by the application process.

(k) Logic Definitions

These definitions are adapted from the ANSI SQL standard X3H2-89-27 of November 16, 1988.

In a single column-predicate:

(K1) Let X denote the result of the first value or column-spec and let y denote the result of the second value or column-spec. The values x and y must be comparable values.

(K2) $(x \text{ bool-op } y)$ is unknown if x and y must be comparable values.

(K3) If x and y are non-null values, $(x \text{ bool-op } y)$ is either true or false:

- $(x = y)$ is true if x and y are equal.
- $(x <> y)$ is true if x and y are not equal.
- $(x < y)$ is true if x is less than y .
- $(x > y)$ is true if x is greater than y .
- $(x <= y)$ is true if x is not greater than y .
- $(x >= y)$ is true if x is not less than y .
- $(x != y)$ is true if x and y are not equal.

(K4) Numbers are compared with respect to their algebraic value.

(K5) The comparison of two character strings is determined by the comparison of <characters> with the same ordinality. If the character strings do not have the same length, the comparison is made with a temporary copy of the shorter character which has been extended on the right with space characters so that it has the same length as the other character strings.

(K6) Two character strings are equal if all <characters> with same ordinality are equal. If two character strings are not equal, their relation is determined by the comparison of the first pair of unequal <characters> from the left end of the character strings. This comparison is made with respect to implementor-defined collating sequence.

(K7) Although $(x = y)$ is unknown if both x and y are null values, a null value is identical to or is a duplicate of another null value.

(K8) $(x \text{ IS NULL})$ is either true or false.

(K9) $(x \text{ IS NULL})$ is true if and only if x is the null value.

(K10) $(x \text{ IS NOT NULL})$ has the same result as $\text{NOT}(x \text{ IS NULL})$.

(K11) $\text{NOT}(\text{true})$ is false, $\text{NOT}(\text{false})$ is true, and $\text{NOT}(\text{unknown})$ is unknown. AND and OR are defined by the following truth tables:

AND	true	false	unknown
true	true	false	unknown
false	false	false	false
unknown	unknown	false	unknown

OR	true	false	unknown
true	true	true	true
false	true	false	unknown
unknown	true	unknown	unknown

(K12) As implied by the syntax, expressions within parentheses are evaluated first and when the order of evaluation is not specified by parenthesis, NOT is applied before AND, AND is applied before OR, and operators having the same precedence level are applied from left to right.

(l) Mapping Rules for Select

The SQL precompiler will choose a secondary copy of data for retrieval if the requesting process is on the same host and if the CDMA has permitted it through the use of the host and of the ALLOW RETRIEVAL clause of the CREATE MAP command. If DISALLOW RETRIEVAL has been specified, which is also the default, only the primary copy of data will be retrieved. A description of the CREATE MAP command may be found in the NDDL User's Guide.

(m) Mapping Rules for precompiler Generated Referential Integrity Checks

The SQL precompiler will select the primary copy of data.

(n) Union, Minus, and Intersect

The following rules have been derived or extracted from the ANSI SQL standard X3H2-89-27 of November 16, 1988.

(N1) The selections within parentheses are evaluated first, and when the order of evaluation is not specified by parentheses, INTERSECT is applied before UNION or MINUS and the set-operators at the same precedence level are applied from left to right.

(N2) Each selection is evaluated and stored in temporary tables.

(N3) Let T and T' denote tables. The result of T set-operator T' is a table R, derived as follows:

Case:

- (N4) If the set-operator is UNION, then:
 - a. Initialize R to an empty table.
 - b. Insert each row of T and each row T' into R.
- (N5) If the set-operator is INTERSECT, then:
 - a. Initialize R to an empty table.
 - b. For each row of T, if a duplicate of that row exists in T', insert that row of T into R.
- (N6) If the set-operator is MINUS, then:
 - a. Initialize R to an empty table.
 - b. Insert each row of T into R.
 - c. For each row of R, if a duplicate of that row exists in T', eliminate that row from R.
- (N7) T and T' must have the same number of columns. Corresponding columns in T and T' must have identical data type descriptions.
- (N8) The degree of R is the same as the degree of T and T'.

3.3 Delete Command

3.3.1 Syntax

The DELETE command removes rows from an external-schema table. The DELETE command has the following syntax:

```
DELETE FROM table-name  
[WHERE { predicate-spec }]
```

where

table-name is defined for the relational view,

predicate-spec is either a column-, join-, between-, or null-predicate (see Appendix A).

3.3.2 Comments

(a) Locking

A DELETE command inside a transaction places a "key lock" on deleted rows until a COMMIT command is encountered. This lock ensures that another process cannot insert a row with the key of the deleted row until the DELETE action has been finalized by a COMMIT or ROLLBACK command. Actual lock mechanisms depend on the internal-schema databases.

(b) WHERE Clause

The WHERE clause is used to specify which rows qualify to be deleted. For selective qualification of rows, the WHERE clause has the same power of expression as it does in a SELECT statement.

The qualifications specified in the WHERE clause of an SQL statement will be "ANDed" with those specified in the WHERE clause of the CREATE view. These qualifications include the column to value predicates and may be expressed within nested parentheses. This parenthesized logic will be enforced by the precompiler. If a DELETE has no WHERE clause, but there is a WHERE clause on the view, the view WHERE clause will still be enforced.

Some columns cannot be specified in a WHERE clause because the column in the conceptual schema maps to non-normalized database structures in the internal-schema databases. In particular, a conceptual-schema column that maps to a data field in a repeating group in the internal database will not have a unique value for each row. The precompiler will recognize this problem and reject the SQL request. The CDM Administrator should inform the user of these restrictions before precompilation. The CDMA can determine these by examining the conceptual-internal schema mapping relationship.

(c) Mapping Restrictions

The external-schema table (your view of the table) must map to one complete conceptual-schema entity class. This means that a request to DELETE a row in a table in your view can be rejected by the system because other information that you are not (necessarily) aware of would also have to be deleted in the conceptual-schema representation of the database. Thus, it may be necessary to determine the conceptual-schema structure and mapping to external views to formulate a correct DELETE command to explicitly delete all the columns of a row in the conceptual schema.

The entity class (in the conceptual schema) may map to just part (or all) of one or more record types in the actual database (in the internal schema). If just part of a record type is mapped to, that deleted part is filled with null-values and the remainder is left as is. The null values used are those specified in the NDDL DEFINE DATABASE command.

The SQL precompiler will generate delete transactions for secondary copies if the CDMA has permitted it through the use of the ALLOW UPDATE clause of the CREATE MAP command. If DISALLOW UPDATE has been specified, which is the default, only the primary copy will be deleted. The delete of these secondary copies are not guaranteed.

(d) Integrity Constraints

A request to delete a conceptual-schema entity that has dependent entities will be rejected at runtime. Those dependent entities cannot be ignored; their existence depends on the existence of the independent entity.

(e) Null Values

The specification of internal-schema null-values is DBMS dependent. If a DELETE statement does result in filling part of a record with null-values, the actual null-values used will be those values specified in the CDM.

(f) Examples:

```
DELETE FROM OFFER
  WHERE STATUS = 'EXPIRED'
```

```
DELETE FROM OFFER
```

```
DELETE FROM OFFER
  WHERE STATUS = 'OLD'
  AND DATE < :CUT-DATE
  AND TYPE != 'RETRO'
```

3.4 Insert Command

3.4.1 Syntax

The INSERT command adds rows to an external-schema table. The INSERT command has the following syntax:

```
INSERT INTO table-name [(column-name, ...)]
VALUES (value, ...)
```

where

column-name is defined for the relational view,

value is a numeric variable, a character variable, a number, or a character string.

3.4.2 Comments

(a) Locking

An INSERT command places an EXCLUSIVE lock automatically until a COMMIT or ROLLBACK command is encountered. Actual lock mechanisms depend on the internal-schema databases.

(b) Specified Columns

The columns of the table are specified by their column-names. The values are related to the columns by their respective orders of appearance. If the column-names are omitted, then the values are related to the columns by the order of the columns as defined in the relational view.

(c) Value and Variable Input

The list of items enclosed in parentheses on the VALUES clause can contain values and/or program variables for input. The data types of values explicitly given must agree with the data type of target columns. Values cannot be calculated by an arithmetic expression within the INSERT statement. Refer to Appendix B for proper variable definitions.

(d) Mapping Restrictions

The external-schema table (your view of the table) must map to one complete conceptual schema entity class. This means that a request to INSERT a row in a table in your view can be rejected by the system. Thus, it may be necessary to determine the conceptual-schema structure and mapping to external views to formulate a correct INSERT command to explicitly insert all the columns of a row in the conceptual schema.

The entity (conceptual schema) may map to just part (or all) of one or more internal-schema (actual databases) record types. If just part of a record type is "mapped to," that part not inserted is filled with null-values. Moreover, if a record type in the internal database maps to two conceptual-schema entities, inserting in one conceptual entity, followed by the other, will result in two partial record instances in the internal database, rather than one complete instance; the precompiler does not view this result as incorrect and will not issue a rejection or warning.

The SQL precompiler will generate insert transactions for secondary copies if the CDMA has permitted it through the use of the ALLOW UPDATE clause of the CREATE MAP command. If DISALLOW UPDATE has been specified, which is the default, only the primary copy will be updated. The insertion of these secondary copies are not guaranteed.

(e) Integrity Constraints

A request to insert a conceptual-schema entity that is dependent in a relation but for which no independent entity exists will be rejected at runtime. A dependent entity cannot exist without its associated independent entities, one for each relation in which it is dependent.

A request to insert a conceptual-schema entity with key value equal to that of an entity already in the database will be rejected at runtime. Key values must be unique.

(f) Null Values

The specification of internal-schema null-values is DBMS dependent. If an INSERT statement does result in filling part of a record with null-values, the actual null-values used will be those values specified in the CDM.

(g) WHERE clause of CREATE VIEW

It is permissible to insert rows not in the external view. Therefore, the qualifications in the WHERE clause of the CREATE VIEW will have no significance for SQL processing of an INSERT.

(h) Examples:

Note: If no column-names are specified, the values of columns to be inserted must be in ordinal position of columns as defined in the NDDL CREATE VIEW command.

```
INSERT INTO DEPT
VALUES (:DNO, :DNAME, :DLOC, :DSIZE)
```

```
INSERT INTO DEPT (DEPT_NO, DEPT_NAME, DEPT_LODE, DEPT_SIZE)
VALUES (12, 'ENGR', 'B1', 'SMALL')
```

```
INSERT INTO DEPT
VALUES (:DEPT-NUM, :DEPT-NAME, 'B1', :DEPT-SIZE)
```

3.5 Update Command

3.5.1 Syntax

The UPDATE command changes values in an external-schema table. The UPDATE command has the following syntax:

```
UPDATE table-name
SET column-name = value, ...
[WHERE { predicate-spec    }]
```

where

table-name and column-name are defined for the relational view,

value is a numeric variable, or a character variable, or a number, or a character string

predicate-spec is either a column-, join-, between-, or a null-predicate (see Appendix A).

The columns to be changed and the values to be entered must be explicitly specified in the SET clause.

3.5.2 Comments

(a) Integrity Constraints and Mapping Restrictions

Three specific integrity constraints are enforced by the system. First, the UPDATE command cannot be used to change the values of a column that corresponds to the key of an entity in the conceptual schema. Thus, some requests that have an apparently correct syntax might be rejected. To modify a key, it is necessary to first DELETE and then INSERT the entity. Second, referential integrity is enforced. If a foreign key is to be modified, there must exist a parent for the new key. Third, it is not permissible to change just part of a foreign key; the entire foreign key must be changed.

Some columns cannot be updated alone because the column in the conceptual schema maps to non-normalized database structures in the internal-schema databases. In particular, a conceptual-schema column that maps to a data field in a repeating group in the internal database will not have a

unique value for each row. The precompiler will recognize this problem and reject the SQL request. You can determine these restrictions before precompilation only by examining conceptual-internal schema mapping relationships.

The SQL precompiler will generate update transactions for secondary copies if the CDMA has permitted it through the use of the ALLOW UPDATE clause of the CREATE MAP command. If DISALLOW UPDATE has been specified, which is the default, only the primary copy will be updated. The update of these secondary copies are not guaranteed.

(b) Locking

An UPDATE command within a transaction places an EXCLUSIVE lock automatically until a COMMIT or ROLLBACK command is encountered. Actual lock mechanisms depend on the internal-schema databases.

(c) SET Clause

The SET clause specifies the new values that are to be given to values in designated columns. The new value can be contained in a program variable or be given explicitly. New values cannot be calculated by arithmetic expressions in the UPDATE command, nor can they be contained in a structure.

(d) WHERE Clause

The WHERE clause is used to specify which rows qualify to be changed. For selective qualification of rows, the WHERE clause has the same power of expression as it does in a SELECT statement.

The qualifications specified in the WHERE clause of an SQL statement will be "ANDed" with those specified in the WHERE clause of the CREATE view. These qualifications include the column to value predicates and may be expressed within nested parentheses. This parenthesized logic will be enforced by the precompiler. If an UPDATE statement has no WHERE, but there is a WHERE clause on the view, the view WHERE clause will still be enforced.

Note: It is permissible to update rows in the view that are moved thereby out of the view.

Some columns cannot be specified in a WHERE clause because the column in the conceptual schema maps to non-normalized database structures in the internal-schema databases. In particular, a conceptual-schema column that maps to a data field in a repeating group in the internal database will not have a unique value for each row. The precompiler will recognize this problem and reject the SQL request. The user can determine these restrictions before precompilation only by examining conceptual-internal schema mapping relationships.

(e) Examples:

```
UPDATE OFFER
  SET STATUS = 'EXPIRED'
  WHERE DATE < :CUTDATE
```

```
UPDATE OFFER
  SET RESPONSIBLE-DEPT = 'BENEFITS'
```

```
UPDATE DEPT
  SET STATUS = 'INACTIVE',
      LOC = 'INACTIVE',
      RESPONSIBLE-MNGR = :MNGR-INPUT
  WHERE DNO = :DEPT-NO-INPUT
```

3.6 Transaction Commands

3.6.1 Rollback Command

This SQL command causes the system to undo any actions accomplished since the current execution began or the last COMMIT or ROLLBACK command was issued. The databases will be returned to their previous states and all locks will be removed.

The syntax is: ROLLBACK WORK

3.6.2 Commit Command

The COMMIT command causes all actions to become permanent since the current execution began or the last COMMIT or ROLLBACK command was issued. All existing locks on records will be removed.

The syntax is COMMIT WORK

3.7 Cursor Commands

For queries which return more than one row of data, a cursor must be used with the SELECT statement.

The cursor is a table (work area) used by the SELECT statement. One cursor is associated with one SELECT statement and may be executed repeatedly for different variations of a SELECT statement.

The cursor is created when a transaction is initiated and destroyed when that transaction is terminated.

These are four commands associated with cursors:

DECLARE CURSOR

OPEN CURSOR

FETCH

CLOSE CURSOR

3.7.1 Declare Cursor Command

The DECLARE CURSOR statement names a cursor and specifies the SELECT statement to be used.

Refer to Section 3.1 for syntax and semantics of this statement.

3.7.2 Open Cursor Command

The OPEN CURSOR statement designates a work area to be used, evaluates the SELECT statement defined in the DECLARE CURSOR command, and establishes an active set of rows to satisfy the SELECT statement.

The syntax is:

OPEN cursor-name

cursor-name is a name that has been previously defined in a DECLARE CURSOR command within the host program.

3.7.2.1 Comments

(a) Location of command

An OPEN CURSOR statement may not physically appear in a host program before the corresponding DECLARE CURSOR command.

(b) Cursor Status

A cursor must be in a closed state when an OPEN CURSOR command is issued. By default, when a cursor is declared, it is placed in a closed state.

3.7.3 Fetch Command

The FETCH command reads the rows in the active set established by the OPEN CURSOR command and identifies the variable names which will contain the results of the SELECT statement.

The FETCH command advances the position of an open cursor to the next row of the cursor's ordering and retrieves the values of the columns of that row into the program variable names.

The syntax is:

```
FETCH cursor-name INTO [:]variable-name [[ [:] INDICATOR] ind-var-  
name],...]
```

cursor-name is a name that has been previously defined in a DECLARE CURSOR command within the host program.

variable-name and ind-var-name are variables defined with the BEGIN DECLARE SECTION and END DECLARE SECTION of the host program.

3.7.3.1 Comments

(a) Location of command

A FETCH command may not physically appear in a host program before the corresponding DECLARE CURSOR command.

(b) Cursor Status

A cursor must be in an opened state when a FETCH command is issued.

(c) When a FETCH command has positioned the cursor after the last row, or no rows were retrieved at all, the value 100 will be assigned to the SQLCODE parameter.

3.7.4 Close Cursor Command

The CLOSE CURSOR command destroys the resources of the active set established with the OPEN CURSOR command.

The syntax is:

```
CLOSE cursor-name
```

cursor-name is a name that has been previously defined in a DECLARE CURSOR command within the host program.

3.7.4.1. Comments

(a) Location of command

A CLOSE CURSOR command may not physically appear in a host program before the corresponding DECLARE CURSOR command.

(b) Cursor Status

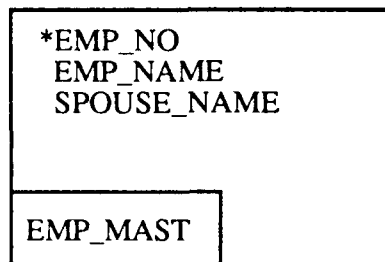
A cursor must be in an opened state when a CLOSE CURSOR command is issued. By default, when a cursor is declared, it is placed in a closed state.

3.8 Distributed Update Restrictions

For examples 1 through 5, assume that the CDMA has "allowed" update for entity EMP-MAST. Also, there is a 1 to 1 mapping for AUCs EMP-NAME and SPOUSE-NAME, whereas the keyed AUC EMP-NO is mapped for preference 1 to database:1 and mapped for preference 2 to database:2

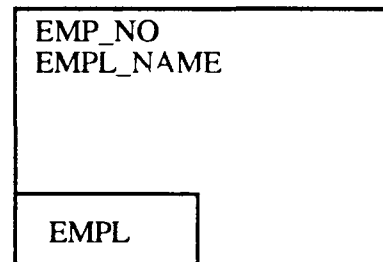
Conceptual Schema

(Assume EXTERNAL SCHEMA
is the same)

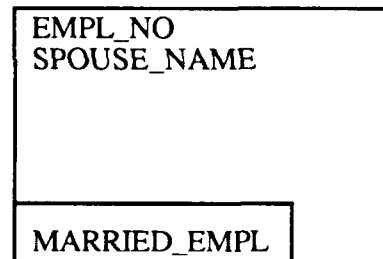


Internal Schema

DATABASE: 1



DATABASE: 2



SQL Transaction Examples:

1. INSERT INTO EMP_MAST
(EMP-NO, EMP-NAME, SPOUSE-NAME) VALUES
(100, 'MR X', 'MRS X')
2. UPDATE EMP_MAST
SET SPOUSE_NAME = 'NEW MRS X'
WHERE EMP_NO = 100
3. UPDATE EMP_MAST
SET SPOUSE_NAME = 'NEW MRS X'
WHERE EMP_NAME = 'MR X'
4. DELETE FROM EMP_MAST
WHERE EMP_NO = 100

5. DELETE FROM EMP_MAST
 WHERE EMP_NAME = 'MR X' OR
 SPOUSE_NAME = 'MRS X'

Example 1:

Two subtransactions will be generated to insert into records EMPL of Database:1 and MARRIED_EMPL of Database: 2.

There are no restrictions for "insert" Actions. All copies or sources will be updated.

Example 2:

1 subtransaction will be generated to modify record MARRIED_EMPL of Database:2 with the appropriate qualifications (i.e., where EMPL_NO = 100)

Example 3:

This SQL request will be rejected because the record we are attempting to update (MARRIED_EMPL of Database:2) does not contain the relevant qualification (i.e., EMP_NAME = 'MR X').

Example 4:

Two subtransactions will be generated to delete records EMPL of Database:1 and MARRIED_EMPL of Database:2. The CS to IS Transformer Module (where CS is conceptual schema and IS is internal schema) will recognize that the qualification criteria is present in all the records being deleted.

Example 5:

This SQL Delete will fail because both the qualifications are not present in both the records being deleted.

3.9 Error Codes

Codes of interest to the SQL application are:

100	no data
0	no error
< 0	error
-49901	failure of a referential integrity test on an insert or modify
-49902	failure of a referential integrity test on a delete
-49903	failure of a key uniqueness test on an insert
-44306	failure of a domain verification

This error code will be found in the variable SQLCODE after every SQL statement. This is a variable generated into the user program.

Note: A DELETE or UPDATE of an empty set is not considered as an error.

The SQLCODE variable is defined differently depending on the language of the host program. The following names and definitions apply:

<u>LANGUAGE</u>	<u>NAME</u>	<u>DEFINITION</u>
COBOL	SQLCODE	S9(9) COMP
FORTRAN	SQLCOD	INTEGER
C	sqlca.sqlcode	long

SECTION 4

EMBEDDING SQL IN A PROGRAM

4.1 COBOL Syntax

The coding syntax for embedding an SQL statement in a COBOL program is:

```
EXEC SQL
      SQL statement
END-EXEC.
```

4.2 FORTRAN Syntax

The coding syntax for embedding an SQL statement in a FORTRAN program is:

```
(COL 6) EXEC SQL
      SQL statement
```

4.3 C Syntax

The coding syntax for embedding an SQL statement in a C program is:

```
EXEC SQL
      SQL statement
;
```

APPENDIX A

BACKUS-NORMAL FORM OF SQL

SQL-command	::= select-command insert-command delete-command update-command commit-command rollback-command declare-cursor-command close-command open-command fetch-command WHENEVER SQL-condition-exception-action embedded-SQL-declare-section-command
select-command	::= SELECT [DISTINCT] {expr-spec,...} [ALL] {*} INTO { :variable-name [[INDICATOR] =ind-var- name],... } { parameter-name [[INDICATOR] parameter- name],... } FROM table-name [table-label],... [WHERE predicate-spec [AND predicate-spec ...]]
insert-command	::= INSERT INTO table-name (column-list) VALUES (source-list)
delete-command	::= DELETE FROM table-name [table-label] [WHERE predicate-list]
update-command	::= UPDATE table-name [table-label] SET column-assgmt-list [WHERE predicate-list]
commit-command	::= COMMIT WORK
rollback-command	::= ROLLBACK WORK
declare-cursor-command	::= DECLARE cursor-name CURSOR FOR query-expression [order-by-clause]
close-command	::= CLOSE cursor-name
open-command	::= OPEN cursor-name

<code>fetch-command</code>	<code>::= FETCH cursor-name INTO</code> <code>{ :variable-name [[INDICATOR] =ind-var-</code> <code>name],... }</code> <code>{ parameter-name [[INDICATOR] ind-parm-</code> <code>name],... }</code>
<code>embedded-SQL-declare-section-command</code>	<code>::= EXEC SQL BEGIN DECLARE SECTION</code> <code>SQL variable-name</code> <code>EXEC SQL END DECLARE SECTION</code>
<code>between-predicate</code>	<code>::= column-spec [NOT] BETWEEN</code> <code>{column-spec value} AND</code> <code>{column-spec value}</code>
<code>bool-op</code>	<code>::= = != > >= < <= <></code>
<code>boolean-factor</code>	<code>::= [NOT] boolean-primary</code>
<code>boolean-primary</code>	<code>::= predicate-spec (predicate-list)</code>
<code>boolean-term</code>	<code>::= boolean-factor</code> <code> boolean-term AND boolean-factor</code>
<code>column-assignment-list</code>	<code>::= column-assignment-spec </code> <code>column-assignment-list,</code> <code>column-assignment spec</code>
<code>column-assignment-spec</code>	<code>::= column-spec = value</code>
<code>column-list</code>	<code>::= column-spec column-list, column-spec</code>
<code>column-predicate</code>	<code>::= column-spec bool-op value value</code> <code>bool-op column-spec</code>
<code>column-spec</code>	<code>::= column-name table-name.column-</code> <code>name table-label.column-name</code>
<code>digit</code>	<code>::= 0 1 2 3 4 5 6 7 8 9</code>
<code>direction</code>	<code>::= ASC DESC</code>
<code>embedded-SQL-command</code>	<code>::= SQL-prefix SQL-command SQL-terminator</code>
<code>exception-action</code>	<code>::= CONTINUE </code> <code>GOTO label</code>
<code>expr-list</code>	<code>::= expr-spec expr-list, expr-spec</code>
<code>expr-spec</code>	<code>::= column-spec function ([DISTINCT]</code> <code>column-spec) [ALL </code>
<code>function</code>	<code>::= AVG MAX MIN SUM COUNT</code>

function-list	::= function-spec function-list, function-spec
function-spec	::= scalar-variable = function
join-op	::= = !=
join-predicate	::= column-spec join-op column-spec [outer-join-op]
label	::= identifier: identifier. unsigned.integer
like-predicate	::= [NOT] LIKE value.specification
null-predicate	::= column-spec IS [NOT] NULL
order-by-clause	::= ORDER BY {unsigned integer} [direction],... {column-spec}
order-spec-list	::= column-spec [direction] order-spec-list, column-spec [direction]
outer-join-op	::= column-spec = column-spec (+)
predicate-list	::= boolean-term predicate-list {OR XOR} boolean-term
predicate-spec	::= join-predicate between-predicate like-predicate null-predicate
query-expression	::= query-term query expression {UNION} [ALL] query term {MINUS} {INTERSECT}
query-term	::= SELECT [DISTINCT] {expr-spec,...} {* } FROM table-name [table-label],... [WHERE predicate-spec [AND predicate-spec ...]]
quoted-variable	::= 'literal-string'
scalar-variable	::= :variable-name [(subscript-list)]
set-operator	::= UNION INTERSECT MINUS
source-list	::= (value-list)

SQL-condition	::= SQLERROR NOT FOUND
SQL-prefix	::= EXEC SQL
SQL-terminator	::= (language dependent; see EMBEDDED SQL User Manual)
SQL-variable-name *	::= C-variable-name FORTRAN-variable-name COBOL-variable-name

*NOTE: See Appendix B for valid variable data types.

subscript-list	::= integer subscript-list, integer
table-list	::= table-name [table-label] table- list, table-name [table-label]
unsigned integer	::= digit ...
value	::= scalar-variable quoted-variable unsigned integer
value-list	::= value value-list, value

APPENDIX B

EXTERNAL SCHEMA DATA TYPES

This appendix consists of the External Schema data types available for COBOL, C, and FORTRAN programmers.

An application programmer writing a COBOL program may define:

a Character variable	(C)	as	PIC	X(n)
a Signed number	(S)	as	PIC	S9(n) V9(n) SIGN LEADING SEPARATE
an Unsigned number	(N)	as	PIC	9(n) v9(n)
a Packed value	(P)	as	PIC	S9(n) V9(n) COMP -3
a Variable character variable	(V)	as	PIC	X(n)
a Decimal number	(D)	as	PIC	S9(n) V9(n) SIGN LEADING SEPARATE
a Numeric number	(N)	as	PIC	S9(n) V9(n) SIGN LEADING SEPARATE

An application programmer writing a FORTRAN program may define:

a Character variable	(C)	as	Character *n
an Integer number	(I)	as	Integer
a Floating point number	(F)	as	Double Precision
a Real number	(R)	as	Real *4
a Variable character variable	(V)	as	Character *n
a Small Integer number	(A)	as	Integer
a Double Precision number	(O)	as	Double Precision

An application programmer writing a C program may define:

a Character variable	(C)	as	char (n)
a Variable character variable	(V)	as	char (n)
an Integer number	(I)	as	long
a Small Integer number	(A)	as	long
a Double Precision number	(O)	as	double
a Floating point number	(F)	as	double